File Cy:

BRL R 1731

BRL

REPORT NO. 1731

SLIP FOR THE BRLESC II COMPUTER

Morton A. Hirschberg

July 1974

USA BALLISTIC RESEARCH LABORATORIES
ABERDEEN PROVING GROUND, MARYLAND

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER BRL Report No. 1731	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* SLIP FOR THE BRLESC II COMPUTER		5. TYPE OF REPORT & PERIOD COVERED Final
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR*(s)* Morton A. Hirschberg		8. CONTRACT OR GRANT NUMBER*(s)*
9. PERFORMING ORGANIZATION NAME AND ADDRESS USA Ballistic Research Laboratories Aberdeen Proving Ground, Maryland 21005		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U.S. Army Materiel Command 5001 Eisenhower Avenue Alexandria, VA 22304		12. REPORT DATE JULY 1974 13. NUMBER OF PAGES 32
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Symmetric List Processing	List Structures	Lists
List Processing	Ring Structures	Reentrant Subprogramming
String Processing	Strings	Rings
Symbolic Processing	Linked Lists	Circular Lists
Recursive Subprogramming	Character Manipulation	

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

SLIP, an acronym for Symmetric List Processor, is a list processing system which carries a forward and backward link as well as a datum. It is symmetric in the sense that lists do not have a preferred orientation; operations which can be carried out on the top of a list can be as easily carried out on the bottom of the list.

List processing languages are formal mathematical languages and have been used in symbolic processing in calculus, circuit theory, mathematical logic, artificial intelligence, and numerous other applications.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

I. INTRODUCTION

This paper describes the SLIP system for the BRLESC II computer. SLIP is an acronym standing for Symmetric List Processor, a collection of subprograms originally designed and written by Professor Joseph Weizenbaum in 1963.

List processors are designed to process data consisting of lists of symbols. List processing has been used for symbolic processing in differential and integral calculus, circuit theory, mathematical logic, translation of natural language, artificial intelligence, and numerous other applications.

SLIP is not an autonomous system, but a set of subroutines embedded within the FORTRAN language. To use SLIP one need only familiarize himself with FORTRAN and the SLIP subprograms. SLIP is highly modular; a call to one subprogram may evoke a dozen other subprograms. A version of SLIP resides on the BRL disc and may be accessed via the * COMPILE card as follows:

```
* COMPILE DISC, SLIP, ALL
```

SLIP is a list processing system in which each list cell (2 FORTRAN words) carries a forward and backward link as well as a datum. It is symmetric in the sense that lists do not have a preferred orientation. That is, operations which can be carried out on the top of a list can be as easily carried out on the bottom of the list.

SLIP (as many other list processing languages) contains the following features: dynamic storage allocation, recursive subprogramming, manipulation of complex data structures, and pushdown stores.

Dynamic storage allocation (2.)* is the automatic assignment and release of storage during execution. The user creates lists as needed and erases them when he is through working with them.

A recursive subprogram is one which calls itself. Generally, a routine may not call itself because the return linkages (where control next passes) would be destroyed. SLIP however provides mechanisms for saving return linkages in a pushdown list (or stack) so control always passes to the proper portion of the program. The recursive feature of SLIP is not as sophisticated as mechanisms provided in other list processing languages.

The data structures of SLIP may contain symbolic as well as numeric information. Information is carried by the relational structure as well as the symbolic content of the data.

---

*Numbers in brackets indicate references.

A pushdown store or stack may be thought of as a store with the property that only the first element of the data may be accessed. Elements are added to the store by "pushing down" the existing list and placing the new element on the stack. When the first element is removed the stack is "popped-up" and the next element becomes available by becoming the first element.

The remainder of this report contains a description of SLIP (for further information see 1. and 3.). In addition, a brief description of the more than 100 subprograms of SLIP is given. Finally, an illustrative example using many but not all of the SLIP features is shown (see 1. for 15 examples which exercise 70% of the SLIP system).

II. THE FUNDAMENTALS OF SLIP

A. SLIP-Cells

The basic unit of SLIP is the SLIP-cell. A SLIP-cell consists of two contiguous computer words (Figure 1). The first word is divided into three fields: a two-bit identifier field (ID), and two address sized fields called the left link (LNKL) and right link (LNKR). The LNKL and LNKR fields contain the addresses of neighboring SLIP-cells. The LNKL field contains the address of the word to the left of it (the word above it if thought of as a stack). The LNKR field contains the address of the word to the right of (or below) it. The machine addresses are called "pointers", and one usually says the LNKL field "points" to the word to the left of (or above) it.

In general, the second word of the SLIP-cell contains a datum. The datum assumes two forms: a datum proper or the above subdivision into ID, LNKL, and LNKR fields.

The ID field of the SLIP-cell identifies the cell as follows:

ID	CELL TYPE
0	List cell with non-name as datum
1	List cell with list name as datum
2	Header cell
3	Reader cell

If the ID is 0, the second word of the SLIP-cell contains a datum proper; if 1, the name of a list; if 2, the name of an auxiliary list and a count of how many times it is referred to; and if 3, the name of a list and a count of the depth of the list (how deep a sublist it is).

A cell which contains the same machine address in both its LNKL and LNKR fields contains the name of a list. If more than one cell contains the name of the same list, that list is said to have aliases.

The Header and Reader cells are special. Every list has one cell which is its Header. The ID and LNKL fields of the second word of the Header allow special information to be attached to the list it heads. If there are four or fewer classes of lists, a list mark may be used (ID field). If more than four classes exist, one must use Description Lists (LNKL field). The List Mark allows enhanced cross-referencing through complex list structures. It is essentially an ID for a sublist. A Description List is an associate list of a main list rather than a sublist or subordinate list (although it may also function as a sublist).

The Reader cell will be discussed in Section II.E.

B. List Structures

A list structure is a set of lists such that all but one of the lists are sublists of the set. Figure 2 shows a simple list structure. A list is a sublist if its name is in the datum field of a SLIP-cell (on a super list), and that cell contains a 1 in its ID field.

Much of the power of list processing derives from being able to create and manipulate list structures. That is, putting the name of one list onto another list and then working with the entire structure as a single entity.

C. List of Available Space

The list of available space (LAVS) is the space in which all the SLIP-cells are formed. This area must be declared by the user in his program. It is best to declare LAVS to occupy all of core memory remaining after the compilation of all FORTRAN programs needed by the SLIP system. Currently, this system requires 11000 words of storage. LAVS is formed by the user calling subroutine INITAS which links together the available space.

In addition to calling INITAS, the user must define the following blank common area:

```
COMMON AVSL, X(100)
```

AVSL is a single FORTRAN word so constructed that its LNKR field contains the address of the first word pair of LAVS and its LNKL field the address of the last word pair. When a SLIP-cell is taken or returned to LAVS, AVSL is modified to reflect this change. In some SLIP systems, AVSL is a list just like any other SLIP list. At BRL, only one directional pointers appear in AVSL.

X is an array containing 100 public lists with aliases X(1),X(2),..., X(100). By public, the aliases are known to all subprograms through the common statement. An important use of public lists is in recursions where they serve in the communication of parameters. The user must not

include a blank common of his own while using the SLIP system.

With SLIP, it is the users responsibility to erase lists when they are no longer needed. This space would then be returned to LAVS. There is no automatic erasure or garbage collection in SLIP.

D. Recursions

One of the most powerful techniques of list processing is recursion. This is the feature which allows a subprogram to call itself. The SLIP subroutines which make recursion possible are VISIT and TERM.

A symbolic label defined by means of the FORTRAN ASSIGN statement is one of the arguments in VISIT's calling sequence. After stacking the proper return address on an internally kept local list, VISIT transfers control to the address provided by the symbolic label.

TERM passes control back to VISIT with a value which will become the value of VISIT. The normal FORTRAN control sequence is executed after normal exit from VISIT.

The statements:

```
      Y = VISIT (GOTO, NOUSE)
        .
        .
        .
      RETURN
GOTO ———————
        .
        .
        .
      CALL TERM (C, NOUSE, TEMP)
```

transfers control to GOTO, executes some code, and returns to VISIT from TERM where Y obtains the value C. Upon exiting from VISIT, normal flow continues (i.e. the RETURN statement is ultimately executed).

One might notice that TERM, a FORTRAN function, has 2 arguments shown in SECTION III.F. and 3 arguments above. The third argument arises at the BRL on BRLESC II when a function is called as a subroutine. In this instance a word must be provided to accommodate the value of the function. TEMP is such a word.

It is possible to go through loops within loops. That is, code may exist containing many calls to VISIT in a given block. The user may look upon the VISIT function as a GO TO operation under control of a DO loop and follow the FORTRAN rules for such operations in using VISIT.

VISIT and TERM have been written independently of the SLIP system and stand alone. Up to 400 recursions have been provided for.

E. The Reader and Advance Functions

The Reader is an indexing apparatus. It allows the user to traverse list structures comprehensively. The Reader is a one-dimensional stack whose elements refer to branch points at which descents were made into sublists. In other words, the Reader keeps a historical record of its traversal through lists.

The advance functions modify the state of the Reader. Advancement may be linear or structural. Linear advancement is an advance through a list element by element and is halted when a Header is reached. Structural advancement is an advance through a list allowing descents into sublists (if advancement is to the left). The next unit of an advance may be a datum, a sublist name (Header), or either of these. Advancement may occur to the left or to the right. This means that we may search through a list from bottom to top (left), or top to bottom (right).

F. Naming Conventions

In general, SLIP disregards the mode of subprogram arguments. The returned value of a function does however follow the standard FORTRAN naming conventions. Names beginning with I, J, K, L, M, or N are considered integers. Names beginning with any other letter of the alphabet are treated as floating point. In order to avoid unwanted conversions, SLIP has two functions, REEL and INTGER, to adjust the value of ill-named functions.

III. THE SUBPROGRAMS OF SLIP

This section gives a brief description of each SLIP subprogram. Wherever possible, the subprograms are grouped along functional lines. Most of the SLIP subprograms are function subprograms and are written in FORTRAN. An indication (in parenthesis) will be given when there is a deviation from this. The codes, Weizenbaum or Hirschberg, will be indicated by a W or H in parenthesis preceding the subprogram name.

It is often desirable to CALL a function. If this is done, the user must append on extra argument to the FORTRAN calling sequence when using the BRLESC II computer. This extra argument contains the value of the function upon leaving the function.

In referring to the argument lists of subprograms, the following naming conventions adopted by Findler (1.) is employed here:

X is a FORTRAN variable
LST is the alias name of a list
MADR is a FORTRAN variable containing a machine address
KADR is a FORTRAN variable containing the machine address of another word
NRD is the alias name of a Reader

A. List Creation

(SUBROUTINE, W) INITAS(X,N) where

X is a linear array of N words. INITAS creates and links together the list of available space. It also creates 100 public lists. The user must call INITAS before any SLIP subprograms can be used.

(W) LIST(LST)

LIST creates an empty list with alias LST. If the value of LST is the literal 9, a local list is created. Local lists are for temporary use.

B. Manipulating List Data

(W) NXTLFT(X,MADR)
(W) NXTRGT(X,MADR)

These functions insert a new cell to the left or right, respectively, of the cell with machine address MADR. The contents of X are placed into the second word of the cell.

(W) NEWTOP(X,LST)
(W) NEWBOT(X,LST)

These functions add a new cell to the top and bottom, respectively, of the list with alias LST. The contents of X are placed into the second word of the cell.

(W) SUBSTP(X,LST)
(W) SUBSBT(X,LST)

These functions replace the contents of the top and bottom cell, respectively, of the list with alias LST with the contents of X.

(W) SUBST(X,MADR)

Function SUBST replaces the contents of the second word of the SLIP-cell with machine address MADR with the contents of X.

(W) INLSTL(LST,MADR)
(W) INLSTR(LST,MADR)

These functions leave the Header of list LST as an empty list and insert the remainder of the list to the left or right, respectively, of the cell with machine address MADR.

(W) TOP(LST)
(W) BOT(LST)

These functions deliver the contents of the top and bottom cell, respectively, of the list with alias LST.

(W) POPTOP(LST)
(W) POPBOT(LST)

These functions deliver the contents of the top and bottom cell, respectively, of the list with alias LST. That cell is then deleted (taken off of the list).

C. Manipulating Cell Data

(ASSEMBLY SUBROUTINE H) SETDIR(I,NL,NR,X)
(ASSEMBLY SUBROUTINE H) SETIND(I,NL,NR,KADR)

These routines store the contents of I, NL, and NR in the ID, LNKL, and LNKR fields of word X or the word whose address is given by KADR. If any of I, NL, or NR has the value -1, the corresponding ID, LNKL, or LNKR field is left unchanged.

(ASSEMBLY H) STRDIR(X1,X2)
(ASSEMBLY H) STRIND(X1,KADR)

These functions store the contents of X1 in X2 or the word whose address is given by KADR.

(ASSEMBLY H) ID(X)
(ASSEMBLY H) LNKL(X)
(ASSEMBLY H) LNKR(X)

These functions deliver the ID, LNKL, or LNKR field of X.

(ASSEMBLY H) CONT(KADR)
(ASSEMBLY H) INHALT(KADR)

These functions return the contents of the word whose address is in KADR. The different names avoid the FORTRAN naming convention.

(H) MADOV(X)

Function MADOV returns the machine address of variable X. It is just a call to the BRLESC II function LOC.

D. Reader and Advance Functions

(W) LRDROV(LST)

LRDROV appoints a Reader for the list with alias LST.

(W) IRARDR(NRD)

IRARDR erases the Reader stack whose name is NRD.

(W) LPNTR(NRD)

This function returns the location pointer field of Reader NRD.

(W) LOFRDR(NRD)

This function returns the list name field of Reader NRD.

(W) LCNTR(NRD)

This function returns the level counter field of Reader NRD.

(W) REED(NRD)

REED returns the contents of the cell to which the Reader NRD currently points.

(W) LVLRVT(NRD)

This function ascends into a superlist from any position in a list. The Reader NRD points to the cell from which the descent originated.

(W) LVLRV1(NRD)

This function is similar to LVLRVT but ascends only 1 level into a superlist.

(W) INITRD(NRD)

This function makes the Reader NRD point to the Header cell of the current list.

(W) LRDRCP(NRD)

This function makes a copy of the Reader NRD.

(W) LSTPRO(LST,NRD)

This function retrieves the Reader NRD referring to the list with alias LST.

(H) DLSRDR(LST)

This function appoints a Reader to the Description List of list LST.

(W) ADV $\begin{Bmatrix}L\\S\end{Bmatrix}$ $\begin{Bmatrix}E\\N\\W\end{Bmatrix}$ $\begin{Bmatrix}L\\R\end{Bmatrix}$ (NRD, FLAG)

These 12 functions advance the Reader NRD. Advancement is linear or structural; the next unit being a datum E, a sublist name N, or either of these W; advancement is to the left L or to the right R. With linear advancement, FLAG is a test flag which remains 0 as long as an error does not occur, or if a Header is not reached. It is set to -1 otherwise. For structural advancement, FLAG remains 0 until the Header of the main list is reached. At this time, FLAG is set to -1.

(W) ADVSL(NRD,J,K)
(W) ADVSR(NRD,J,K)
(W) ADVLL(NRD,J,K)
(W) ADVLR(NRD,J,K)

These functions do the basic bookkeeping and Reader updating for the Reader NRD. The values of J and K delimit advancement options. The user should rarely have direct need of these routines.

(H) ADNLWR(NRD,N,FLAG)

This function advances the Reader NRD N steps linearly to the right. FLAG remains 0 if an error does not occur, or if a Header is not reached. FLAG is set to -1 otherwise.

(W) SEQRDR(LST)

This function appoints a Sequence Reader for the list with alias LST.

(W) SEQLL(KADR,IFLAG)
(W) SEQLR(KADR,IFLAG)

These functions advance the Sequence header KADR linearly to the left and right, respectively. IFLAG is set to -1, 0, or 1 depending upon whether or not the SLIP-cell contains a datum, sublist name, or a Header cell.

(H) SREED(KADR)

This function delivers the contents of the second word of the SLIP-cell to which the Sequence Reader KADR is currently pointed.

(H) LSPNTR(KADR)

This function returns the machine address of the cell to which the Sequence Reader KADR is currently pointed.

(W) SEQSL(KADR,IFLAG)
(W) SEQSR(KADR,IFLAG)

These functions resemble ADVSEL and ADVSER respectively. They descend into sublists for which KADR is the Sequence Reader. When they reach a terminal sublist (one for which no other sublist exists), they behave as two linear sequence functions at that level. IFLAG is returned as -1 for datum cells, and +1 for Header cells.

(H) ADVDL(KADR,VAL)
(H) ADVDR(KADR,VAL)

These functions advance the Description List Reader KADR left and right, respectively, to the next attribute. VAL is given the attribute value unless no attributes remain, in which case, VAL is set to 0.

E. Character Manipulation

The routines described in this section are general purpose character manipulation routines. They stand alone and may be used to advantage outside of the SLIP system.

(H) CP(M)

This function delivers a mask covering the Mth character position. $1 \leq M \leq 10$. M=1 represents the leftmost character position; M=10 the rightmost character position.

(ASSEMBLY H) SQOUT (MASK,X)

This function extracts that portion of X specified by MASK and delivers it right justified and zero filled. X remains unchanged.

(ASSEMBLY H) SQIN(MASK,DATUM,DEST)

This function replaces the portion of DEST that is covered by MASK with the corresponding number of low-order bits of DATUM.

(ASSEMBLY H) SHIN(N,DATUM,DEST)

This function left shifts DEST by N bit positions and replaces the vacated bits with N low-order bits of DATUM.

(ASSEMBLY H) LANORM(X)

LANORM performs left circular shifts in six bit bytes until the leftmost character of X is non-blank. This function operates on 10 characters or 60 bits.

F. Recursion

(SUBROUTINE W) PRESRV(N)

This subroutine preserves the top cell of the first N ($\leq$ 100) public lists.

(SUBROUTINE W) RESTOR(N)

This subroutine deletes the top cell of the first N ($\leq$ 100) public lists.

(H) PARMT(X,N)

This function places X(I), I=1,N on the top of the first N ($\leq$ 100) public lists.

(W) PARMT2(A,B)

This function places A and B on top of public lists 1 and 2 respectively.

(H) VISIT(GOTO,NOUSE)

This function transfers control to the statements with symbolic label GOTO. NOUSE is either an unused argument or PARMT or PARMT2 depending upon the subprogram usage. The value of VISIT is determined by function TERM.

(H) TERM(X,NOUSE)

This function transfers control back to VISIT. X is the returned value. NOUSE is either an unused argument or RESTOR(N) depending upon the subprogram usage.

VISIT and TERM work in tandem. They can stand alone outside of the SLIP system (with functions INHALT and STRIND). Provision has been made to accommodate up to 400 recursions.

G. Input and Output

In addition to the powerful FORTRAN input and output capabilities, SLIP provides several special purpose input-output subprograms of its own.

(W&H) RDLSTA(Z)

Function RDLSTA reads in and displays a list structure. Lists and sublists are delimited by parentheses; elements may be separated by commas. All 80 columns of the card are read. A list structure may be punched on several consecutive cards. In general, blanks are squeezed out but they may be used to improve readability. Elements longer than 10 characters are truncated on the right, and an error warning given; elements shorter than 10 characters are left justified and blank filled. Characters are converted to their BCD (binary coded decimal) equivalent. A list structure is started with an open parenthesis and terminated after the last closed parenthesis by an asterisk. The dummy argument Z in the calling sequence is not used by the function.

A structure given by

(A(B(C)))* would produce the following output:

```
BEGIN INPUT LIST
(A(B(C)))*
WORD = A
WORD = B
WORD = C
END INPUT LIST - LIST NAME = machine address in repeated format
```

(SUBROUTINE W) PRLSTS(LST,M)

This routine prints the second word of every non-name SLIP-cell of the list with alias LST. M, an integer, defines the output mode as follows:

1 INTEGER
2 ALPHANUMERIC
3 REAL - F FORMAT
4 OCTAL

(H) RITEIT(LST,M,N)

This function prints the second word of every non-name SLIP-cell of the list with alias LST. In addition, the machine address of every element of the list is output in octal. M, an integer, defines the output mode as follows:

1 INTEGER
2 REAL - F FORMAT
3 REAL - E FORMAT
4 OCTAL
5 ALPHANUMERIC

N is a flag controlling the output of repeated sublists. If N is 0, repeated sublists are noted put not printed after the first time. If N is non-zero, repeated sublists are printed every time they are encountered.

H. LIST Marks and Description Lists

(W) MRKLST(N,LST)

MRKLST places N (=0, 1, 2, 3) as the List Mark for the list with alias LST.

(W) MRKLSS(N,LST)

MRLKSS places N = (0, 1, 2, 3) as the List Mark for the entire list structure given by LST.

(W) LSTMRK(LST)

This function delivers the value of the List Mark for the list with alias LST.

(H) IRADLS(LST)

IRADLS erases the Description List of the list with alias LST.

(W) NAMEDL(LST)

This function delivers the Header of the Description List associated with the list whose alias is LST.

(W) LISTAV(LST)

This function creates an empty Description List for the list with alias LST.

(W) MAKEDL(LST1,LST2)

This function causes LST1 to become the Description List of the list with alias LST2.

(W) LDATVL(AT,VAL,LST)

This function adds to or creates the Description List LST. If LST exists AT and VAL, an attribute pair are placed on the list. If LST does not exist, it is created and then AT and VAL are placed on it.

(W) NEWVAL(AT,VAL,LST)

This function searches the Description List of the list with alias LST for the attribute AT. If found, VAL is assigned the value of AT, otherwise, both AT and VAL are added to the list.

(W) NOATVL(AT,LST)

This function removes the attribute AT from the Description List of the list with alias LST.

(W) ITSVAL(AT,LST)

This function delivers the value associated with the attribute AT from the Description List of the list with alias LST.

(W) MTDLST(LST)

This function empties the Description List of the list with alias LST.

(W) MADATR(AT,LST)

This function returns the machine address of the cell on the Description List of the list whose alias is LST and contains the attribute AT.

(SUBROUTINE W) DERROR(LST)

This routine prints the Header of list LST if a Description List of the list with alias LST cannot be found. It is an error routine which terminates the job.

(H) SCHATL(AT,LST)
(H) SCHATR(AT,LST)

These functions search the Description Lists of the list with alias LST for the attribute AT to the left and right, respectively.

I. Miscellaneous Subprograms

(W) DELETE(MADR)

This function erases any list cell with the machine address MADR except a Header cell.

(H) EQUAL(A,B)

This functions test A and B for equality. If they are equal, 0 is returned; otherwise, -1 is returned.

(ASSEMBLY H) INTGER(X)

This function allows X to assume an integer name.

(W) IRALST(LST)

This function returns the list with alias LST to available space. It erases the list.

(W) LISTMT(LST)

This function tests to see if the list with alias LST is empty. If so, 0 is delivered, otherwise, -1 is returned.

(W) LOCT(LST)

This function tests to see whether or not its argument LST is a list alias. If so, the returned value is the alias, otherwise, a message is printed and the program is terminated. The user will rarely ever need to use LOCT directly.

(W) LPURGE(LST)

This function deletes all references to lists from the list structure LST.

(W) LSSCPY(LST)

This function creates a copy of the list with alias LST.

(W) LSTEQL(LST1,LST2)

This function compares list LST1 with list LST2 for equality. If they are indentical in structure and content, the returned value is 0, otherwise, -1 is returned.

(W) MADLFT(MADR)
(W) MADRGT(MADR)

These functions return the machine address of the cell to the left and right, respectively, as specified by the machine address MADR.

(W) MADNTP(LST,N)
(W) MADNBT(LST,N)

These functions return the machine address of the Nth cell from the top and bottom, respectively, of the list with alias LST.

(W) MTLIST(LST)

This function returns all the cells of the list with alias LST to available storage except for the Header cell and the Description List (if one exists).

(W) NAMTST(LST)

This function tests the alias of the list LST. If an alias truly exists, 0 is returned, otherwise, -1 is returned.

(W) NUCELL(Z)

This function obtains a new SLIP-cell from the list of available space. The dummy argument Z is not used.

(W) NULSTL(MADR,LST)
(W) NULSTR(MADR,LST)

These functions split the list with alias LST in two from the left and right, respectively, starting at machine address MADR.

(SUBROUTINE W) RCELL(MADR)

This routine returns the cell with machine address MADR to the list of available space.

(ASSEMBLY H) REEL(L)

This function allows L to assume a real (floating point) name.

IV. A SAMPLE SLIP PROGRAM

This section contains a program example using many of the system features, but not all of its subroutines.

Problem:

Read in a list structure containing part of the hierarchy of the BRL management. Find a particular Laboratory Chief's name and insert the name of another Laboratory Chief ahead of the first name. Print the list structures before and after the modification.

The following pages contain the input data (list structure), a listing of the program, and the program output.

Program Inputs:

The following is a list of the input cards to the sample program.

```
(EICHELBERGER(FRASIER,HOFFMAN(JOHNSON,KOKINAKIS,CUMMINGS(
CANDLAND,HIRSCHBERG,HOLLOWAY,JOHNSON,SCHLEGEL))
MURPHY))*
```

These inputs depict the following list structure:

```
EICHELBERGER
            FRASIER
            HOFFMAN
                  JOHNSON
                  KOKINAKIS
                  CUMMINGS
                        CANDLAND
                        HIRSCHBERG
                        HOLLOWAY
                        JOHNSON
                        SCHLEGEL
            MURPHY
```

Each indentation corresponds to a sublist and reflects a lower level in the BRL hierarchy. Notice one can embed sublists with very little effort. Notice also that Murphy is a part of the sublist (FRASIER, HOFFMAN, MURPHY).

Program listing.

Several items appearing on the program listing require special attention. The user must not use any blank COMMON of his own. The common statement

```
COMMON AVSL,X(100)
```

defines the limits of available space and the public lists. The user must call INITAS which sets up the list of available space.

```
        PROGRAM SLIP

C
C  THIS PROGRAM IS AN ILLUSTRATION OF THE SLIP SYSTEM
C  IT WILL DISPLAY A PART OF THE HIERARCHY OF BRL AND THEN MODIFY IT
C
C  SET UP COMMON AND WORKING STORAGE AREAS
C
        COMMON AVSL,X(100)
        DIMENSION WRKSPC(500)
C
C  INITIALIZE WORKING SPACE
      CALL INITAS(WRKSPC,500)
C  READ IN AND DISPLAY A LIST STRUCTURE
      BRLSTF=RDLSTA(Z)
C  PRINT LIST STRUCTURE
      CALL RITEIT(BRLSTF,5,0)
C  INSERT A NAME OF A LAB CHIEF
      CALL LABCHF(BRLSTF,4HREED,7HHOFFMAN)
```

```
C  PRINT LIST
     CALL PRLSTS(BRLSTF,2)
C  END PROGRAM
     CALL EXIT
     END
     SUBROUTINE LABCHF(LST,SYMB1,SYMB2)

C
C  THIS ROUTINE SEARCHES THE LIST STRUCTURE LST FOR SYMB2 AND INSERTS
C  SYMB1 BEFORE IT
C
C  APPOINT A READER FOR THE LIST STRUCTURE
     LRD=LRDROV(LST)
C  ADVANCE READER
    10 TEST=ADVSER(LRD,FLAG)
C  TEST FLAG
     IF (FLAG.NE.0.0) GO TO20
C  IF ELEMENT IS SYMB2 PUT SYMB1 BEFORE IT
     IF (TEST.EQ.SYMB2) CALL NXTLFT(SYMB1,LPNTR(LRD))
     GO TO 10
C  ERASE READER
    20 CALL IRARDR(LRD,TEMP)
C  END ROUTINE
     RETURN
     END
```

Many of the subprograms of SLIP are function subprograms. They may however be called as if they were subroutine subprograms. When this occurs, an extra argument should be provided for to accommodate the value of the function. The call of IRARDR contains the extra argument TEMP for this purpose. Notice the function subprogram NXTLFT does not have an extra argument to provide for the value of the function. When an extra argument is not provided, an error may occur. Fortunately, in this instance no error occurred.

PROGRAM OUTPUTS

Program outputs consist of a listing of each input card followed by a breakdown of every element on that card. An element can contain only 10 characters. If more are included they are truncated on the right and an error warning is printed. The output beginning with BEGIN INPUT LIST and ending with END INPUT LIST - LIST NAME, etc. is produced by FUNCTION RDLSTA. The output beginning with LIST STRUCTURE WITH HEADER, etc. and ending with END OF LIST STRUCTURE is printed by subroutine RITEIT. The numbers are the octal machine addresses of the listed elements. The output beginning with BEGIN LIST and ending with END LIST is printed by subroutine PRLSTS.

We observe that the element REED has been placed before the element HOFFMAN.

The output from subroutine RITEIT reflects the list structure as it is originally read. The output of subroutine PRLSTS contains the modified list structure.

```
                              BEGIN INPUT LIST

                        (EICHELBERGER(FRASIER,    HOFFMAN(

          WORD=EICHELBERG
MORE THAN 10 CHARACTERS IN WORD
          WORD=FRASIER
          WORD=HOFFMAN
                        JOHNSON,KOKINAKIS,CUMMINGS(
          WORD=JOHNSON
          WORD=KOKINAKIS
          WORD=CUMMINGS
                             CANDLAND,HIRSCHBERG,HOLLOWAY,JOHNSON,SCHLEGEL))
          WORD=CANDLAND
          WORD=HIRSCHBERG
          WORD=HOLLOWAY
          WORD=JOHNSON
          WORD=SCHLEGEL
                        MURPHY))*
          WORD=MURPHY
END INPUT LIST - LIST NAME =         1045200000021124
           LIST STRUCTURE WITH HEADER AT           21124
                                                     21131   EICHELBERG
                                        SUBLIST           21132
                                                     21141   FRASIER
                                                     21143   HOFFMAN
                                        SUBLIST           21144
                                                     21153   JOHNSON
                                                     21155   KOKINAKIS
                                                     21157   CUMMINGS
                                        SUBLIST           21160
                                                     21167   CANDLAND
                                                     21171   HIRSCHBERG
                                                     21173   HOLLOWAY
                                                     21175   JOHNSON
                                                     21177   SCHLEGEL
                                        END OF SUBLIST
                                        END OF SUBLIST
                                                     21201   MURPHY
                                        END OF SUBLIST
                                        END OF SUBLIST STRUCTURE
```

```
BEGIN LIST
    EICHELBERG
    BEGIN SUBLIST
    FRASIER
    REED
    HOFFMAN
    BEGIN SUBLIST
    JOHNSON
    KOKINAKIS
    CUMMINGS
    BEGIN SUBLIST
    CANDLAND
    HIRSCHBERG
    HOLLOWAY
    JOHNSON
    SCHLEGEL
    END SUBLIST
    END SUBLIST
    MURPHY
    END SUBLIST
    END LIST
```

REFERENCES

1. Findler, N.V., Pfaltz, J.L., and Bernstein, H.J., Four High-Level Extensions of FORTRAN IV: SLIP, AMPPL-II, TREETRAN, SYMBOLANG, New York: Spartan Books, 1972, pp1-82.

2. Hirschberg, M.A. "Dynamic Storage Allocation for the BRLESC II Computer," Aberdeen Proving Ground, Maryland, (to be published.)

3. Weizenbaum, J. Symmetric List Processor. Communications of the Association for Computing Machinery, 1963, 6, 524-544.

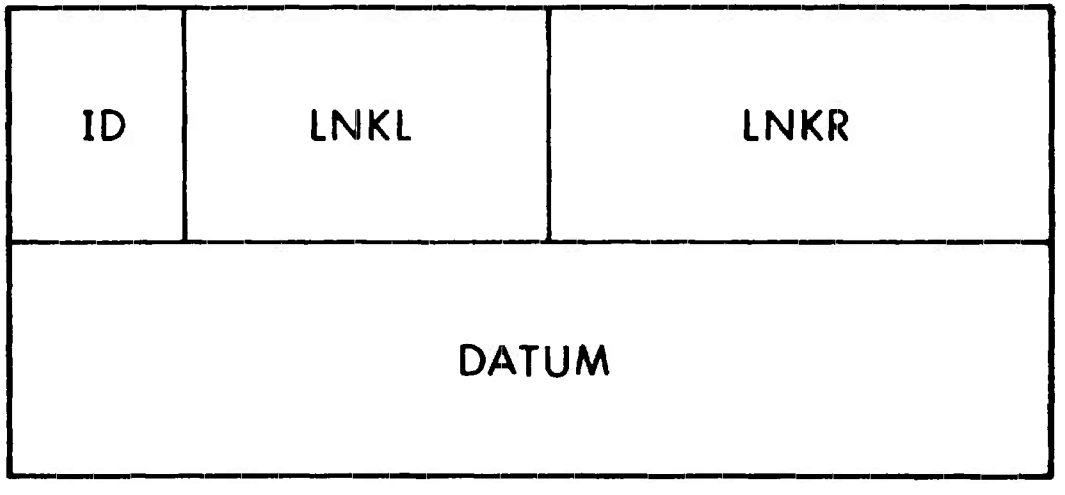

Figure 1. The Format of a SLIP-cell

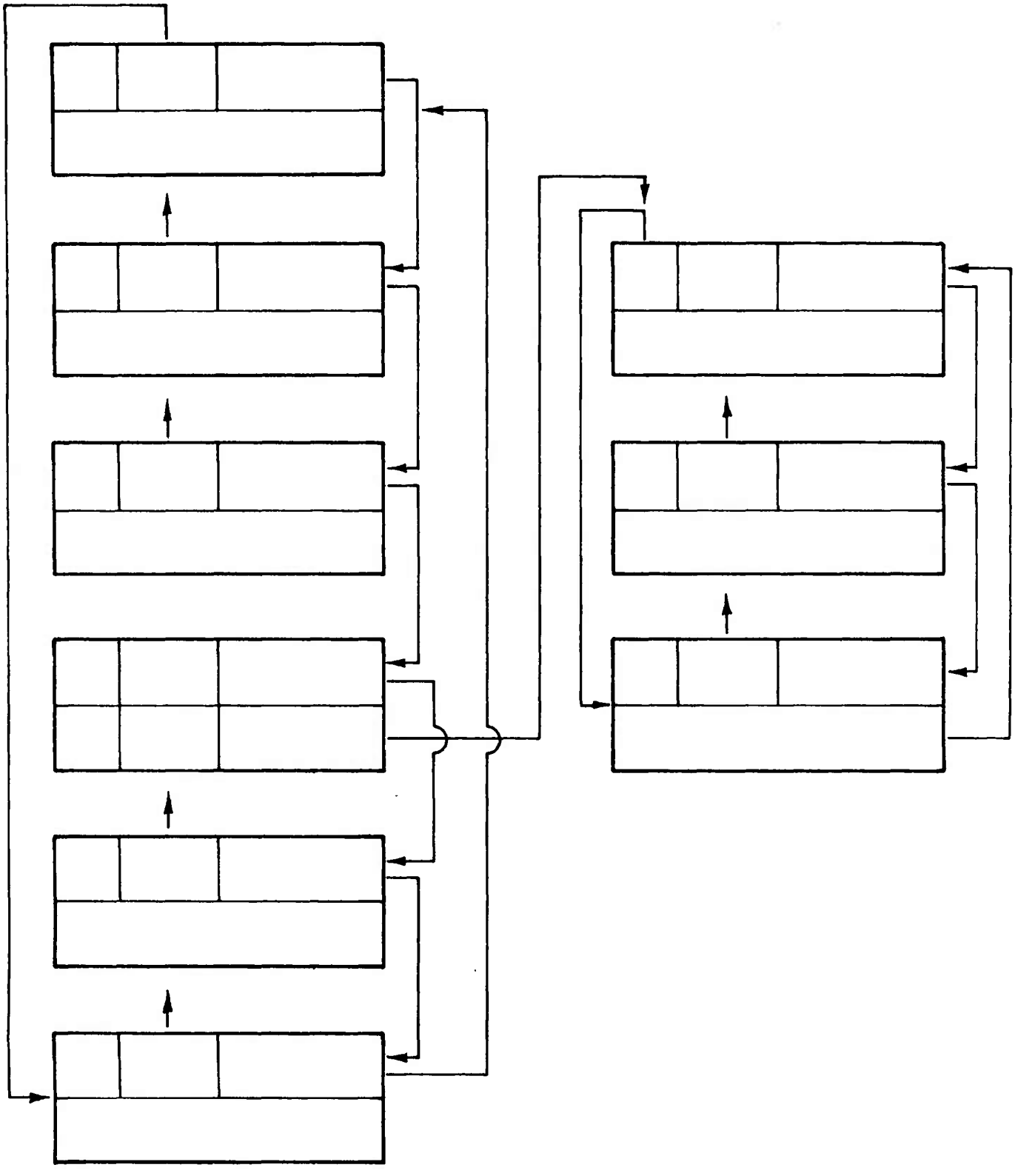

Figure 2- A Simple List Structure Showing a Super List Containing One Sublist

DISTRIBUTION LIST

No. of Copies	Organization
12	Commander Defense Documentation Center ATTN: DDC-TCA Cameron Station Alexandria, Virginia 22314
1	Director Defense Advanced Research Projects Agency ATTN: Tech Info Ctr 1400 Wilson Boulevard Arlington, Virginia 22209
1	Director Institute for Defense Analysis 400 Army-Navy Drive Arlington, Virginia 22202
1	Director Defense Intelligence Agency Washington, DC 20301
1	Commander U.S. Army Materiel Command ATTN: AMCDL 5001 Eisenhower Avenue Alexandria, Virginia 22333
1	Commander U.S. Army Materiel Command ATTN: AMCRD-T 5001 Eisenhower Avenue Alexandria, Virginia 22333
1	Commander U.S. Army Aviation Systems Command ATTN: AMSAV-E 12th & Spruce Streets St. Louis, Missouri 63166
1	Director U.S. Army Air Mobility Research and Development Laboratory Ames Research Center Moffett Field, California 94035
1	Commander U.S. Army Electronics Command ATTN: AMSEL-RD Fort Monmouth, New Jersey 07703
1	Commander U.S. Army Missile Command ATTN: AMSMI-R Redstone Arsenal, Alabama 35809
1	Commander U.S. Army Tank Automotive Command ATTN: AMSTA-RHFL Warren, Michigan 48090
2	Commander U.S. Army Mobility Equipment Research & Development Center ATTN: Tech Docu Cen, Bldg. 315 AMSME-RZT Fort Belvoir, Virginia 22060
1	Commander U.S. Army Armament Command Rock Island, Illinois 61202
1	Commander U.S. Army Harry Diamond Laboratories ATTN: AMXDO-TI Washington, DC 20438
1	Director National Bureau of Standards Department of Commerce Washington, DC 20234

Aberdeen Proving Ground

Dir, USAMSAA
ATTN: AMXSY-D, Dr. J. Sperrazza
Mr. L. Bain
Mr. W. Wenger
Dir, USAHEL
ATTN: Dr. J. Weiss
Dr. R. Bauer
Cmdr, USATECOM